# The Framework for Approximate Queries on Simulation Data

B. Lee[2]*, T. Critchlow[1], G. Abdulla[1], C. Baldwin[1], R. Kamimura[1], R. Musick[3]†, R. Snapp[2], N. Tang[1]

[1]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551

[2]Department of Computer Science, University of Vermont, Burlington, VT 05405

[3]Ikuni, Inc., 3400 Hillview Avenue, Palo Alto, CA 94304

**Abstract**

AQSim is a system intended to enable scientists to query and analyze a large volume of scientific simulation data. The system uses the state of the art in approximate query processing techniques to build a novel framework for progressive data analysis. These techniques are used to define a multi-resolution index, where each node contains multiple models of the data. The benefits of these models are two-fold: 1) they have compact representations, reconstructing only the information relevant to the analysis, and 2) the variety of models capture different aspects of the data which may be of interest to the user but are not readily apparent in their raw form. To be able to deal with the data interactively, AQSim allows the scientist to make an informed tradeoff between query response accuracy and time. In this paper, we present the framework of AQSim with a focus on its architectural design. We also show the results from an initial proof-of-concept prototype developed at LLNL. The presented framework is generic enough to handle more than just simulation data.

**Keywords**: approximate query, multi-model data reduction, multi-resolution index.

*The corresponding author, e-mail: bslee@cs.uvm.edu, fax: (802)656-0696

†Work done while at Lawrence Livermore National Laboratory

# 1 Introduction

Advances in computer technology have created the ability to store and generate data at an unprecedented rate. Unfortunately, the ability to manage and analyze the data has not kept up, and traditional tools cease to be effective as data sets grow from hundreds of gigabytes to terabytes. This poses a severe challenge to data analysis which must address not only the data storage and retrieval costs, which are the focus of projects such as those described in [1], but more importantly must permit the identification and retrieval of the subsets of data that are known to be of interest to the scientists. Furthermore, users need to ask a variety of queries, preferably in an interactive manner, and this warrants an ad-hoc querying tool. Creating such a tool, however, poses a major challenge as the data set size increases, particularly in providing reasonable query processing time.

There has been significant research in approximate queries, which use data proxies generated with data reduction techniques such as those surveyed in [2] to provide approximate answers to queries. Similar to these efforts, AQsim uses models as data proxies, providing a compressed view of the data. Furthermore, as in [8] and [10], AQSim creates a *multi-resolution* index tree in a manner similar to the multi-resolution aggregate (MRA) tree presented by Lazaridis and Mehrotra in [20].

The primary distinction of AQSim from existing efforts is its *multi-model* approach. Because different models are best suited for answering different queries (e.g., a statistical model handles queries requesting the average well, while a spline model easily identifies local minima), this multi-model approach allows us to efficiently handle a wide range of queries. This focus on handling a variety of queries is a necessary contrast to most existing approximate query research which assume only aggregate queries will be asked.

Aimed at handling a broad range of query types, we have designed an architecture that will allow us to explore the various research areas involved in developing such a flexible, scalable system for scientific

data management. A simple version of this architecture has been implemented in the first of what is to be a series of prototypes that will incrementally extend the set of available queries, improve the accuracy of the models, and address scalability issues. These prototypes provide a framework for evaluating new data models, indexing techniques, and mappings between queries and models.

The main contribution of this paper is a description of a system architecture designed to answer queries on large simulation mesh data through approximate query processing, specifically a description of a framework accommodating (1) the creation of multiple models to address a variety of queries, (2) the construction of a multi-resolution index tree to store model data, and (3) the use of the index and models to answer queries. In the next section we present a simplified model of mesh data. Related work is discussed in Section 3. The framework and prototype are described in Section 4, and our current results are presented in Section 5. Finally, conclusion and future work follow in Section 6.

## 2 Simulation Mesh Data

Figure 1 shows an example of Silo [12] mesh data generated by simulation of a can being crushed against a wall, taken midway through a sequence of 44 steps. The data is defined in a three dimensional Euclidean space of variables $x$, $y$, and $z$, and is stored in a sequence of time steps. In this paper, we treat time as a spatial variable. The can's geometrical structure is lying on a cylindrical coordinate system, and its topological structure is a hexahedral grid. There are ten field variables defined on the mesh, all except one of which are node-centered (i.e., defined at a vertex). The other is zone-centered, which corresponds to the center of a volume in 3-dimensional space (or a face in 2-dimensional space) defined by connecting pairs of vertices in a specified order.

Defining a comprehensive mathematical model of mesh data is more complicated than it appears at first

glance because of the variety of mesh types and the ability to combine multiple meshes in the same data set. For example, we may have a regular rectangular mesh, a regular tetrahedral mesh, and an irregular mesh representing different areas of the same data set. For purposes of this paper, we use a simplified view of meshes in which they are described by three abstract data types: a feature set, a mesh topology graph, and a vertex coordinate set. Each data type has an associated query type such as field, topological, and geometrical. [3] Because we anticipate that most queries will be field queries, we further simplify the model by considering only feature sets and ignoring the other data types. Thus, for this paper, a mesh can be viewed as a feature set defined by an ordered set of tuples:

$$\{< e, f_1, f_2, \cdots, f_m >\} \tag{1}$$

where $e$ denotes a mesh element, and $f_i, i = 1, 2, \cdots, m$, denotes the associated field variables. A mesh element can be a vertex, edge, face, volume, or hyper-volume (i.e., 4-dimensional volume).

## 3 Related Work

Data proxies used for approximate queries are in the form of cached statistical summary data [6, 7, 10] or statistically or mathematically compressed models [5, 8]. Cached summary data has been used for statistical databases over a decade [6, 4] and recently for OLAP databases [10, 7]. For example, in [6], statistical summaries computed previously are saved in a table and reused to answer subsequent summary queries, within an acceptable error, without accessing the base data. Bell Lab's AQUA [10, 9] project is a representative example of using cached summary data in the OLAP domain. With sampling and histogram techniques, AQUA precomputes statistical summaries of the contents of data cubes, and uses the summaries to provide approximate answers to primarily aggregate queries [7]. The works by Shatkay and Zdonik [5] and Chakrabarti et al. [8] are examples of using mathematically compressed models. Using Haar

wavelet transformation technique, Shatkay and Zdonik showed approximate queries on time series data, and Chakrabarti et al. on OLAP data. AQUA [9] and the work by Chakrabarti [8], in particular, provide answers at different levels of accuracy depending on the error constraint given by the user, namely support "multi-resolution" queries.

To precompute statistical summaries of OLAP data cubes, AQUA uses key-foreign key relationships to develop a table ordering, then effectively samples the results of multi-table joins, called "join synopses" [11], by sampling the first table in the ordered list. Processing multi-table aggregate queries requires simply rewriting them to use those synopses. When the query set is known in advance, a provably near-optimal set of multidimensional histograms can be selected to precompute an optimal set of aggregates over various data cubes defined by subsets of dimensions. These samples and histogram are maintained incrementally as the base data is updated.

Although similar to AQUA conceptually, AQSim is distinguished in several aspects. First, the data does not map to a non-trivial relational format well because of its high dimensionality, complex attributes, and one-to-one relationships, as exemplified by [13, 14, 15]. Thus, we do not have the key-foreign key relationships that enable the easy creation of join synopsis. This significantly hampers our ability to benefit from aggregate joins. In addition, because we expect queries to occur over almost all of the field and spatial variables, creating indices to retrieve each of these variables would be prohibitively expensive. Finally, the use of sampling in our models is discouraged because the information of most interest to our users is the outliers, which are often lost in sampling.

To use multidimensional wavelets as a model of OLAP data, Chakrabarti, et al. [8] convert relational tables to multivariate histograms and apply the wavelet decomposition to the resulting multidimensional matrix. In addition, they describe how relational operators, such as select, project, join, and aggregation, can be processed using the wavelet coefficients without requiring the values be reconstructed. Their experimental

results show the superiority of wavelets to sampling or histogram, for their data sets. Unfortunately, because of the continuity of mesh field variables, as well as the high variance in the field range, this technique would require binning [16] to be useful in our domain. However, binning has another set of problems that make it inappropriate at this time. For instance, determining the bin size requires repeated accesses to the data, which is infeasible for such a large volume as in our domain.

In [20], Lazaridis and Mehrotra present a generic multi-resolution tree structure and the associated index search and update algorithms. The tree, called a Multi-Resolution Aggregate (MRA) tree, is used as an index for storing, retrieving, and modifying the aggregates of data partition represented by an index node. Assuming the spatial uniformity of data, the algorithms access the index nodes to progressively calculate approximate answers to aggregate queries within user-provided time and error constraints. They consider the typical SQL aggregates, i.e., count, sum, min, max, and avg. Conceptually, our multi-resolution index can be regarded as an instance of the MRA tree resulting from a four-dimensional spatial partitioning. However, the nodes in our multi-resolution index contain the model data (e.g., wavelet coefficients) produced with statistical and mathematical modeling techniques as well as the aggregate information, allowing them to answer more complex queries. In addition, our index tree, used for retrieval only, does not impose the intricate update overhead of aggregates in index nodes as the MRA tree does, thus accommodating the use of more sophisticated aggregates.

# 4 Prototype Architecture

In this section we present the architecture of the current AQSim prototype. The final system will be developed as a series of prototypes.

## 4.1 Design objectives

The AQSim design is founded upon two aspects of its approach to approximate query answering: multi-resolution and multi-model. The fundamental assumption of approximate infrastructures is that not all queries need precise answers. For example, when scientists are performing initial data exploration, getting a fast answer is more important than the answer being completely accurate. To that end, we utilize a *multi-resolution index tree* that allows the user to make a tradeoff between response time and accuracy. This index is created by partitioning the mesh data into multiple levels of granularity and building models of each partition. Queries are then answered using only this index and the associated models. Less accurate responses access only the top of the tree, while the most detailed answers require the nodes found at the leaves of the tree.

An objective of AQSim is the support for multiple models to handle diverse data types and query types in one system. This requires an extensible architecture to incorporate additional modeling techniques. Adding a new modeling technique to this infrastructure involves adding an object that both models the data and regenerates it. This new class inherits, from an abstract class, and implements the modeling and querying operations appropriately.

Figure 2 shows the proposed architecture of AQSim, with modules shown in rectangular boxes, and files in round boxes.

## 4.2 Preprocessing

AQSim, like most traditional approximate query infrastructures, has two phases: preprocessing and query resolution. The preprocessing phase creates the summary or model data, and the query processing phase uses these models to answer queries. The reason for separating these phases is that the modeling is computationally very expensive and should be done only once if queries are to proceed quickly and interactively. Preprocessing also allows for the collection of metadata that describes, for example, which partitions are amenable to which

modeling technique. There are a number of approaches we have considered when designing the preprocessor, including whether to use top-down or bottom-up partitioning, whether to use fixed or adaptive modeling complexity (e.g., the number of coefficients), and when to terminate the partitioning. These options are briefly discussed in the remainder of this section.

The bottom-up approach starts with small partitions and combines them in a well-defined way. This approach can be very efficient for those models (e.g., statistics, wavelets) that allow combining the coefficients from the smaller partitions to obtain the coefficients for the larger partitions, without generating them from scratch for every partition. This efficiency is expected to offset additional partitions that are generated and discarded because their combined partition is still modeled more accurately than required by a user. However, because this incremental generation of model coefficients is not possible for many models, and the implementation of a bottom-up partitioning algorithm is complicated, we are not pursuing a bottom-up approach at this time.

The top-down approach currently used in AQSim is a 4-way octree-like partitioning. It repeatedly partitions the data until the model error for a partition falls below a user-specified threshold. For a partition with an error above the threshold, we can either (1) divide it into subpartitions and model each subpartition using the same modeling complexity or (2) increase the modeling complexity until the model meets the error requirement or reaches a maximum complexity. The fixed complexity approach has been chosen over the adaptive approach because of its simplicity. In the top-down approach, the nodes at or near the root of the index tree may have modeling errors too large to ever be retrieved as the result of index search, and thus may be regarded as wasted storage space. However, because there are relatively few of these nodes (due to the large branching factor of the tree), we do not expect this to be significant problem for most data sets.

We have a choice of how to define the partition error over multiple models, which is used to decide whether to partition further or not. For example, we could use the (a) minimum, (b) average, or (c) maximum

of the model errors. Figure 3 compares the nodes generated using these three cases in a simple example. For option (a), the partitioning stops when at least one model meets the error requirement; for (c), all models; and for (b), about half. Consequently, the tree height, construction time, and storage space are minimized for option (a) and maximized for option (c). Conversely, in the case of (a), only one or at best a small number of nodes can be used to reconstruct the data within the error limits, whereas, in the case of (c), any model can be used, thus allowing for the choice of the best model for a given query type. Option (c) is a compromise between the two, and is what we use.

Given our adoption of top-down partitioning with fixed model complexity and termination based on the average modeling error, the preprocessing phase shown in Figure 2 works as follows. It starts by converting a mesh file from a simulation-specific format to a consistent representation. Then, the Index Constructor reads the data and passes it to the Mesh Partitioner to create a multi-resolution index tree and associated models. The resulting tree is written to an index file and a set of model files. Mesh partitioning is done top down using a four-dimensional bisection that equally divides the $x$, $y$, $z$, and time variables in the current data set. Each non-null (sub)partition is represented by a node in the tree, which contains multiple models of the data in that partition and links to its children. In addition to the current index tree, traditional database indexing can be applied to one or more field variables in the data set. This would allow us to quickly identify the nodes matching the query on the field values. The critical question is which indices should be generated given a set of field variables since they should be only for frequently executed queries. Since we do not have insight into common queries, we are delaying the implementation of this feature until we have an active set of users.

We currently use a statistical model as our only model. This model provides the minimum, maximum, mean, and standard deviation of each variable. In addition to the model parameters (i.e., coefficients) for a specific partition, each model also contains an error that reflects the accuracy with which it represents the

original data. We use the mean squared error as the modeling error. We are currently implementing our second model, a univariate wavelet transformation [17, 18]. While wavelets have been used to compress mesh data for large-scale visualization problems [19], we are using them in a different way. Because the wavelet coefficients have a unique mathematical interpretation, we can use them to answer certain queries directly. For example, Haar coefficients represent the distance between two points of a given resolution, so by examining them we can answer queries requesting areas with high rates of changes relatively quickly. As this example shows, by querying the models for information implicitly contained in their representation, we are able to extend the set of allowable queries with minimal computational overhead beyond the preprocessing phase.

### 4.3 Query resolution

Queries are initiated through an Oracle Server[1] which directly passes the query statements – specified in a string similar to an SQL "where" clause – to the AQSim Query Engine Data Cartridge through its object-relational interface. We currently support range queries across all variables (e.g., given ranges for a subset of the defined variables, return regions where these restrictions hold). The query statement is parsed by the Query Engine to identify the referenced variables, values, and functions.

The Metadata module, which is currently not implemented, will store information used by both the Index Constructor, when registering a new index, and the Query Engine, when selecting the index and model appropriate to a specific query. It will describe which query types a model handles well and the parameters for estimating query response time. Because our current prototype uses only one index and model per data set, this selection step is omitted.

[1] Oracle is used so that queries sent to this system may be combined with queries to other data stored in the relational database environment.

Once the appropriate model is identified, the Index Searcher uses the index to identify a set of candidate nodes that meet the error requirements and may satisfy the search condition. An index node may satisfy the condition if the ranges of all referenced variables overlap their ranges in the partition associated with the node. For each node, the Data Value Reconstructor (DVR) inside the Index Searcher uses the selected model to regenerate the results. The data returned by the DVR is evaluated by the Predicate Processor to eliminate those outside the ranges in the query condition. This is performed as a separate step because the filtering may require evaluating complex, user-defined functions that we do not want applied to all nodes encountered by the Index Searcher. Figure 4 illustrates the generation and filtration of data points. The gray boxes in Figure 4a denote the partitions associated with the index nodes found by the Index Searcher; the dots in Figure 4b denote the data points generated by the DVR; and the dots in Figure 4c denote the data points that satisfy the query condition, returned by the Predicate Processor.

The desired result is a collection of points that can be visualized. To accomplish this, the Predicate Processor output is sent to the FV2Mesh module, which creates a mesh file. It currently produces a Silo point mesh, which looks like a constellation when visualized. The next prototype is expected to generate zone meshes by using the bounding box information contained in the nodes to define the topology.

# 5 Current Results

We have tested the current prototype with the crushing can data set shown in Figure 5. The figure shows the EQPS ("equivalent plastic strain") field of the *crushing can* mesh data at time steps 10, 20, and 30. Figure 6 shows the same data set represented as a point mesh, for which each node in the original mesh is represented as a single point. This mesh was obtained by executing a query that encompassed the entire data set and constrained the error to zero. Figure 7 shows the results of a query predicate condition "EQPS $>$ 0.578788

and EQPS $<$ 1.44697" at full resolution (i.e., allowed error = 0). Figure 8 shows the same output at a reduced resolution that uses only 65% of the nodes of the full resolution case. The data points are sparser in both time and space for the coarser resolution query.

Although the resulting images are visually not as appealing as the original data image, primarily due to the use of points instead of zones, even the highly compressed representation provides enough visual information to convey significant information about the underlying data. We are planning to improve the resulting visualization by using each partition's four-dimensional bounding box to define a zone and visualize these zones instead of points. Nonetheless, our preliminary results demonstrate the viability of our approach.

# 6 Conclusion and Future Work

In this paper, we gave an overview of the design and status of the AQSim approximate query infrastructure. This system is aimed at enabling users to ask ad-hoc queries over large-scale simulation mesh data. Our design uses models to provide a compressed view of the data, and queries these models as opposed to the data itself. The system uses multiple models to handle a wide range of data and query types. To allow users to trade query response time for accuracy, we use a multi-resolution index to reference the models. The current prototype is a framework in which we intend to explore various research areas in an effort to develop a system that executes approximate, ad-hoc, multi-resolution queries on terabyte scale simulation mesh data.

While we expect the AQSim design will evolve as the project progresses, it is difficult to determine what the impact of future enhancements will be. There is still much work remaining, and we are currently investigating: alternative consistent mesh representations that include topology information; new mesh partitioning and modeling methods; modules to support various forms of query results; and metadata representations.

## Acknowledgment

We thank Edward Smith at LLNL for his comments on the paper. This work was performed under the auspices of the U.S. DOE by LLNL under contract No. W-7405-ENG-48. For UVM authors, this research was partially supported by University of Vermont UCRS Faculty Research Grant No. PSCI00-1.

## References

[1] http://esdis-it.gsfc.nasa.gov/MSST/conf2000/, Online Proceedings of the *Eighth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies* in cooperation with *Seventeenth IEEE Symposium on Mass Storage Systems*, March 27-30, 2000, College Park, MD, USA.

[2] J. Hellerstein (Ed.), Data Engineering Bulletin: Special Issue on Data Reduction Techniques, 20:4 (1997)

[3] B. Lee, R. Snapp, R. Musick, Toward a Query Language on Simulation Mesh Data: an Object-oriented Approach, in: Proc. Intl. Conf. Database Systems for Advanced Applications, 2001, pp. 242-249

[4] M.C. Chen, L.P. McNamee, On the Data Model and Access Method of Summary Data Management, in: IEEE Transactions on Knowledge and Data Engineering 1:4 (1989) pp. 519-529

[5] H. Shatkay, S.B. Zdonik, Approximate Queries and Representations for Large Data Sequences, in: Proc. Intl. Conf. Data Engineering, 1996, pp. 536-545

[6] S. Abad-Mota, Approximate Query Processing with Summary Tables in Statistical Databases, in: Proc. Intl. Conf. Extending Data Base Technology, 1992, pp. 499-515

[7] V. Poosala, V. Ganti, Fast Approximate Query Answering Using Precomputed Statistics, in: Proc. Intl. Conf. Data Engineering, 1999, p. 252

[8] K. Chakrabarti, M.N. Garofalakis, R. Rastogi, K. Shim, Approximate Query Processing Using Wavelets, in: Proc. Intl. Conf. Very Large Database, 2000, pp. 111-122

[9] http://www.bell-labs.com/project/aqua/, AQUA Project Home Page, Lucent Technologies.

[10] S. Acharya, P.B. Gibbons, V. Poosala, S. Ramaswamy, The Aqua Approximate Query Answering System, in: Proc. ACM SIGMOD Intl. Conf. Management of Data, 1999, pp. 574-576

[11] S. Acharya, P.B. Gibbons, V. Poosala, S. Ramaswarmy, Join Synopses for Approximate Query Answering, in: Proc. ACM SIGMOD Intl. Conf. Management of Data, 1999, pp. 275-286

[12] http://www.llnl.gov/meshtv/manuals.html, Silo Documentation Version 4.0, Lawrence Livermore National Laboratory, Livermore, CA

[13] http://nssdc.gsfc.nasa.gov/cdf/cdf_home.html, The Common Data Format, The National Space Science Data Center, Greenbelt, MD

[14] http://www.unidata.ucar.edu/packages/netcdf/index.html, NetCDF, University Corporation for Atmospheric Research

[15] http://hdf.ncsa.uiuc.edu/, The NCSA HDF Home Page, The National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, IL

[16] S. Goil, Harsha, and A. Choudhary, MAFIA: Efficient and Scalable Subspace Clustering for Very Large Datasets, Technical Report CPDC-TR-9906-010, Northwestern University, 1999

[17] A. Antoniadis, G. Oppenheim (Ed.), Wavelets and Statistics, Springer-Verlag, 1995

[18] W. Härdle, et al., Wavelets, Approximation, and Statistical Applications, Springer-Verlag, 1998.

[19] M. Bertram, M. Duchaineau, B. Hamann, K. Joy, Lifted Bicubic Subdivision-Surface Wavelets for Geometry Reconstruction, in: Proc. SIGGRAPH, 2000

[20] I. Lazaridis, S. Mehrota, Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure, in: Proc. ACM SIGMOD Intl. Conf. Management of Data, 2001

# List of Figures


1 Crushing can mesh data. . . . . 17

2 AQSim prototype architecture. . . . . 18

3 Partitioning for different termination schemes. . . . . 19

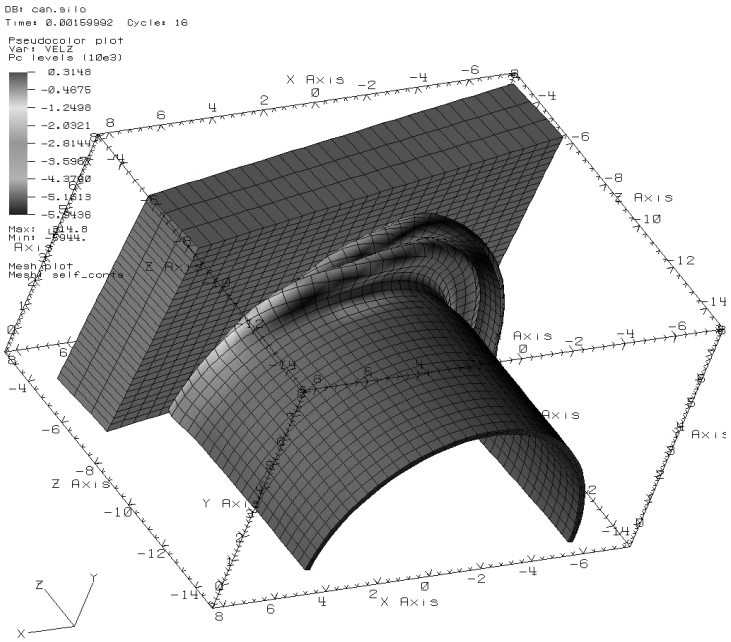
4 Data point generation in overlapping partitions. . . . . 20

5 Crushing can data as a zone-based mesh. . . . . 21

6 Crushing can data as a point mesh. . . . . 22

7 Query output as a point mesh at full resolution. . . . . 23

8 Query output as a point mesh at a reduced resolution. . . . . 24




(Field VELZ at time step 8)

Figure 1: Crushing can mesh data.

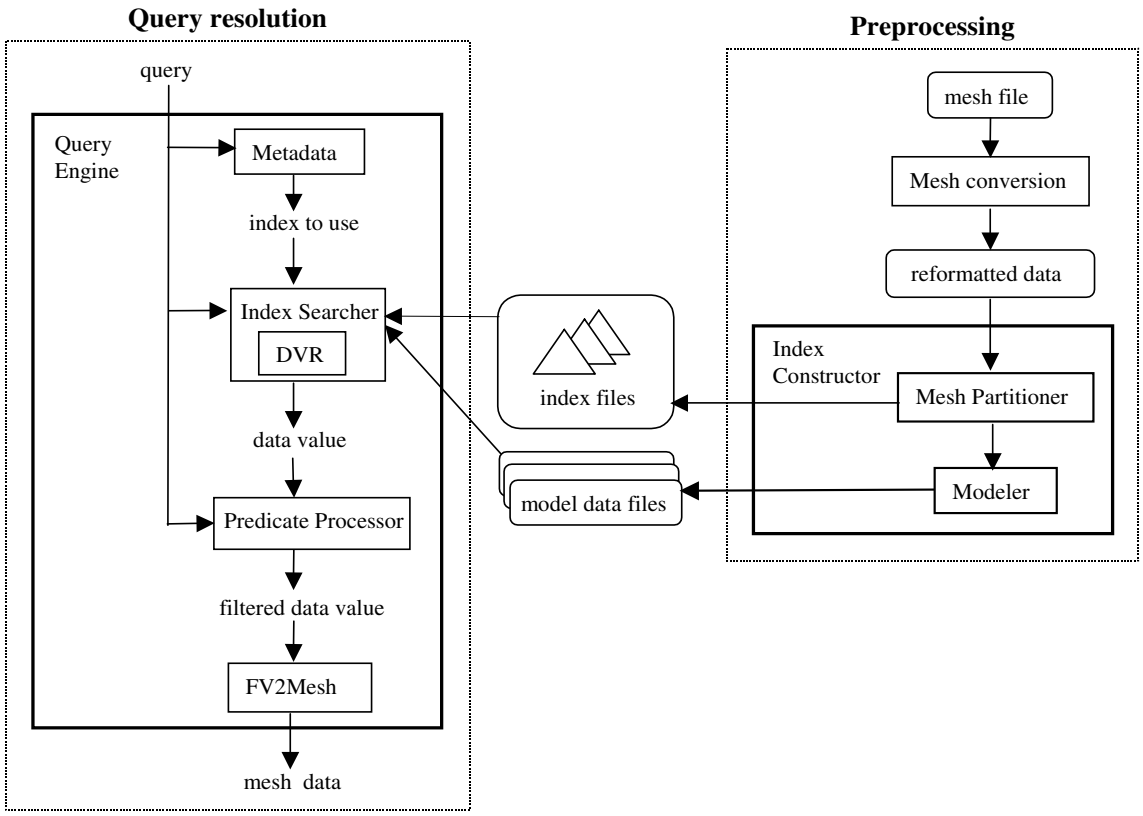



Figure 2: AQSim prototype architecture.

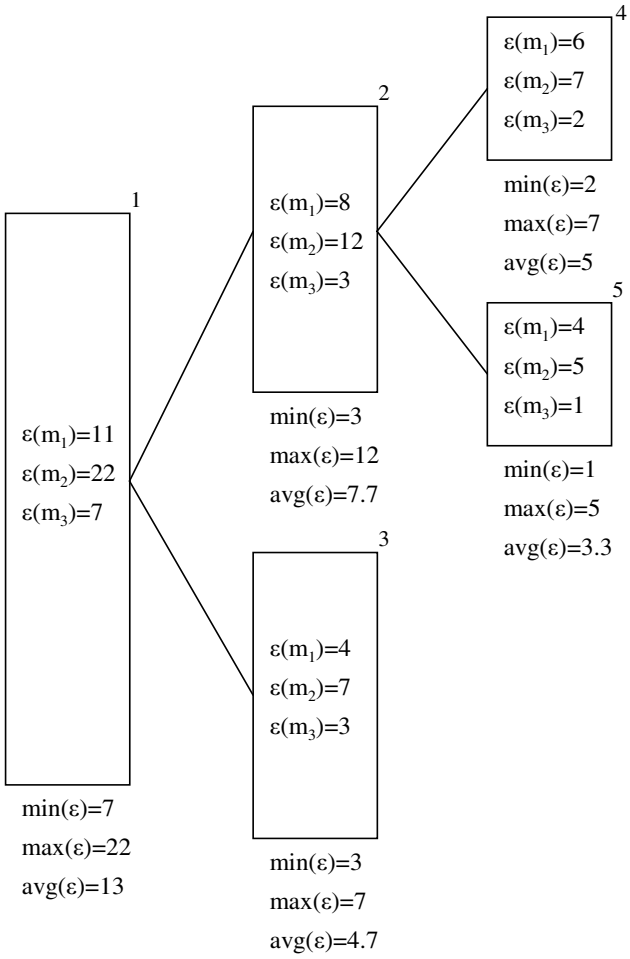
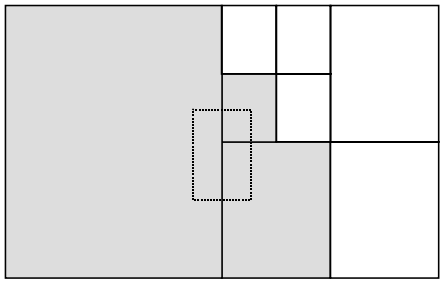
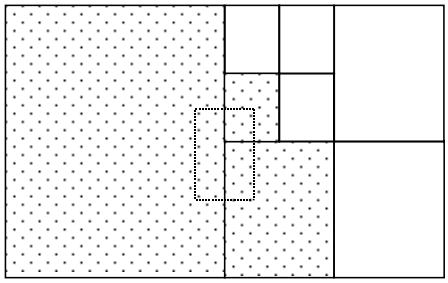


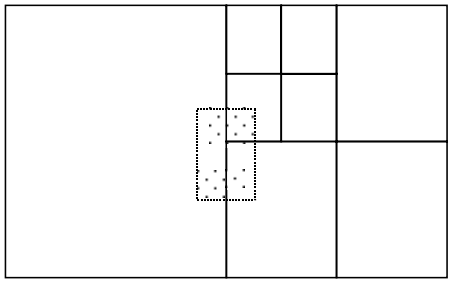

Figure 3: Partitioning for different termination schemes.



(a) Partitions selected by the Index Searcher

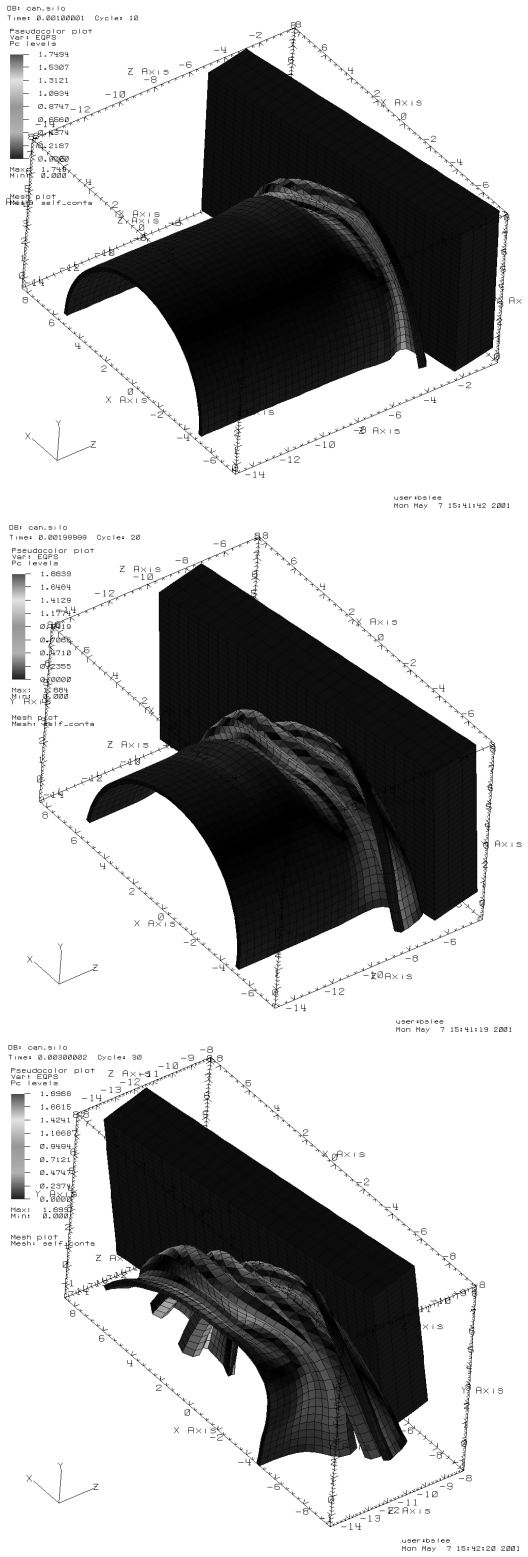


(b) Data points generated by the Data Value Reconstructor inside the Index Searcher



(c) Data points filtered by the Predicate Processor

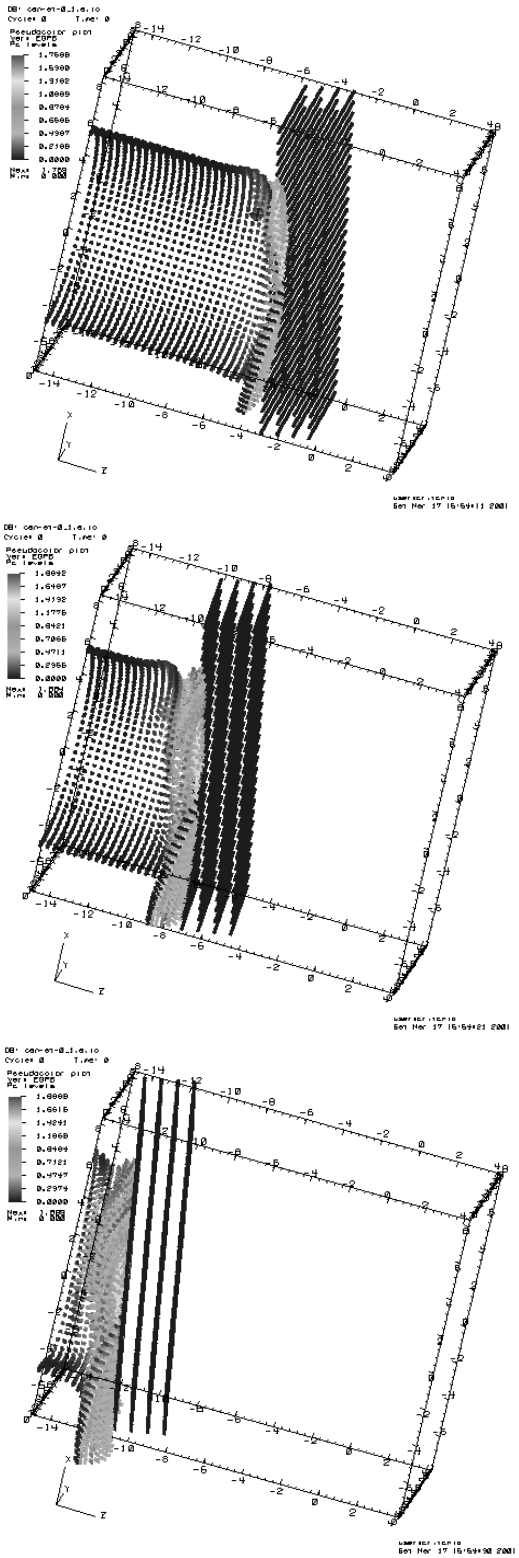
Figure 4: Data point generation in overlapping partitions.



(Field EQPS at time steps 10, 20, and 30.)



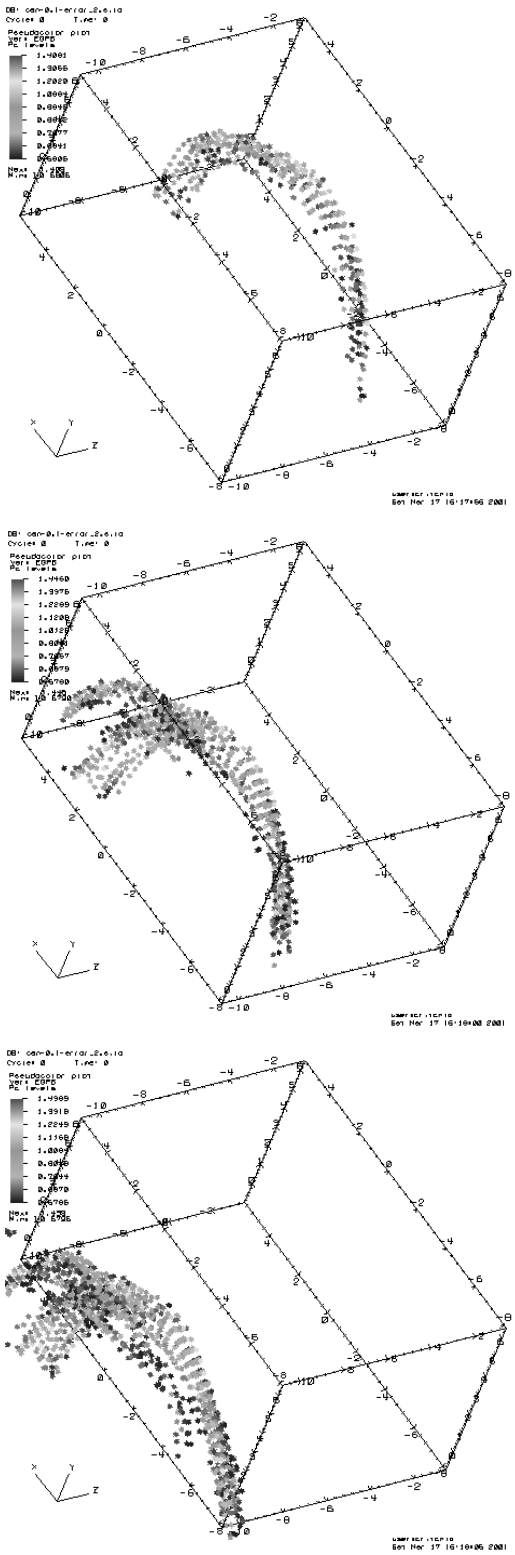
Figure 5: Crushing can data as a zone-based mesh.



(Field EQPS at time steps 10, 20, and 30.)

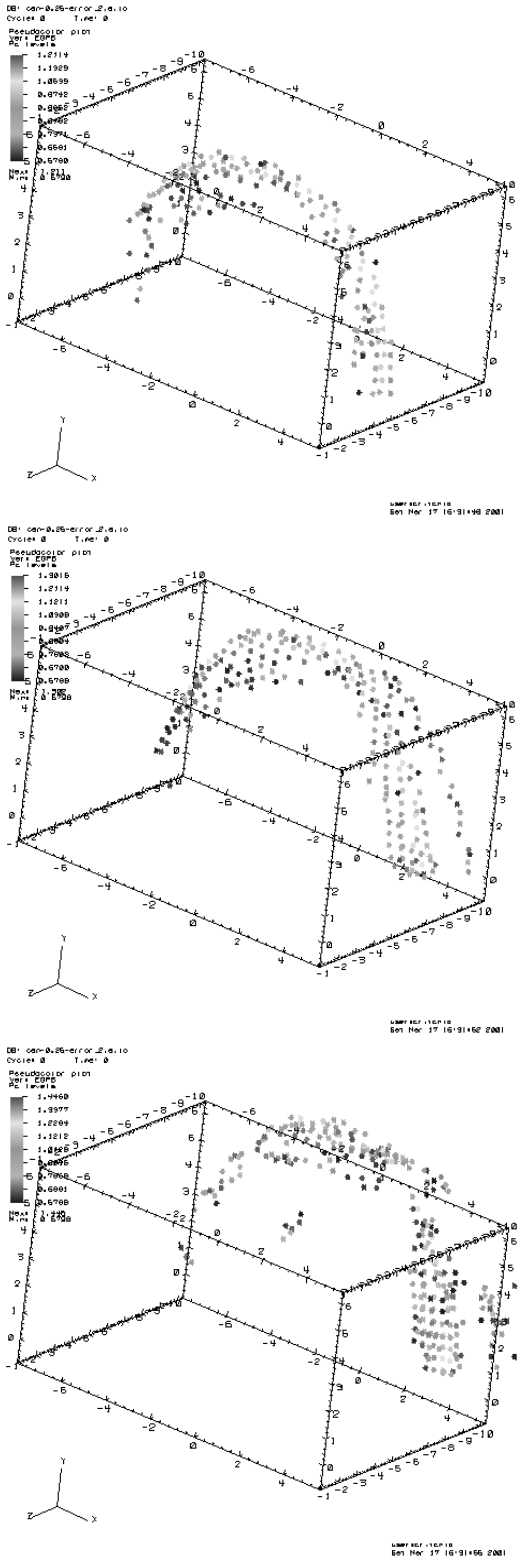


Figure 6: Crushing can data as a point mesh.



(Field EQPS in the range 0.578788 $\sim$ 1.44697 at time steps 10, 20, and 30.)

Figure 7: Query output as a point mesh at full resolution.



(Field EQPS in the range 0.578788 $\sim$ 1.44697 at time steps 10, 20, and 30.)

Figure 8: Query output as a point mesh at a reduced resolution.